Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

|  |  |
| --- | --- |
|  |  |

**Альбом звітів лабораторних робіт**

з дисципліни "Алгоритми та структури даних"

Виконав: студент групи КН

Перевірив: старший викладач

Мошко Є.О.

Харків 2022

Зміст

[Хеш таблиця 3](#_Toc119234674)

[Блок схеми методів 4](#_Toc119234675)

[Робота програми 7](#_Toc119234676)

[Аналіз 7](#_Toc119234677)

[Хеш таблиця ланцюжками 8](#_Toc119234678)

[Блок схеми методів 9](#_Toc119234679)

[Робота програми 13](#_Toc119234680)

[Аналіз 13](#_Toc119234681)

[Хеш таблиця з відкритою адресацією 14](#_Toc119234682)

[Блок схеми методів 15](#_Toc119234683)

[Робота програми 19](#_Toc119234684)

[Висновки 21](#_Toc119234685)

# Хеш таблиця

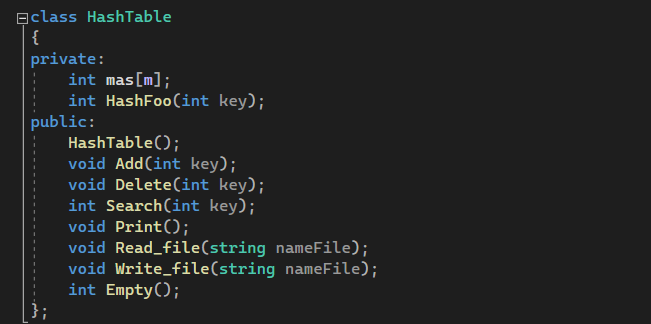
**Хеш-таблиця** - це структура даних, яка реалізовує інтерфейс - *асоціативний масив*, а саме, вона дозволяє зберігати пари (ключ, значення) і здійснювати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення за ключем.

Існує два типи хеш таблиць:

З ланцюжками;

З відкритою адресацією.

1. Нехай розмір хеш-таблиці дорівнює m = 20, а хеш-функція має вигляд: h(k) = [m(kA mod 1)], где А = ( 5 - l)/2

Для хеш таблиці створив клас HashTable з потрібними методами додавання, видалення, пошуку елементів та роботою з файлом.

## Блок схеми методів

Функція додавання елемента та пошуку хеш значення.

void HashTable::Add(int key)

{

mas[HashFoo(key)] = key;

}

int HashTable::HashFoo(int key) {

double rez = (key \* ((sqrt(5) - 1) / 2));

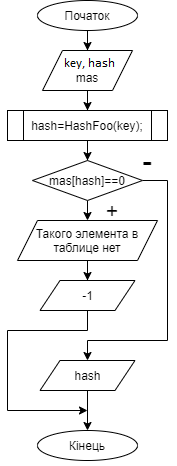
double fraction, integer;

fraction = modf(rez, &integer);

int h = (int)m \* fraction;

return h;

}



Функція пошуку ключа

int HashTable::Search(int key)

{

if (mas[HashFoo(key)] == 0)

{

cout << endl << "Такого элемента в таблице нет" << endl;

return -1;

}

else

{

return HashFoo(key);

}

}



Функція видалення ключа

void HashTable::Delete(int key)

{

if (mas[HashFoo(key)] == 0)

{

cout << endl << "Такого элемента в таблице нет" << endl;

}

else

{

cout << endl << "Элемент таблицы успешно удален" << endl;

mas[HashFoo(key)] = 0;

}

}

## Робота програми

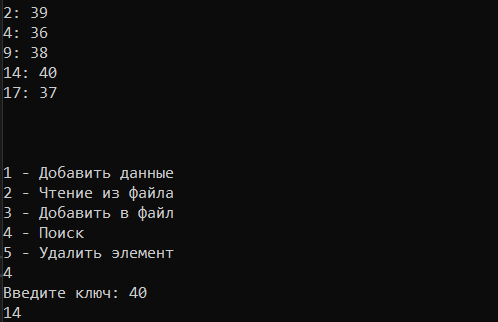
Запуск програми за допомогою файлу з ключами свого вариінту.

Рис. 1 – пошук елементу

Рис. 2 – видалення елементу 40

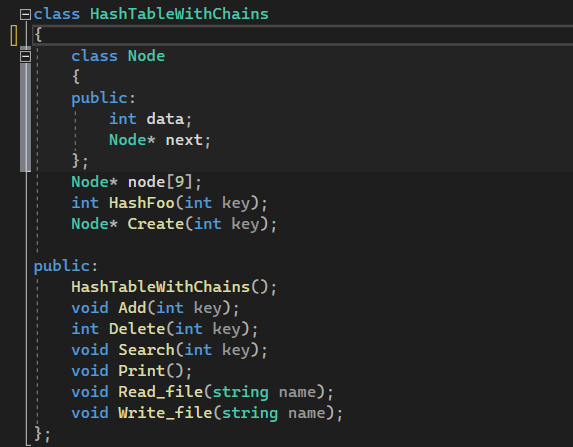
## Аналіз

Так як даний вид хеш-таблиці передбачає те що в дану таблицю будуть записані лише такі дані котрі не будуть викликати колізію то тимчасова складність для функції додавання, пошуку, видалення у всіх випадках буде – .

# Хеш таблиця ланцюжками

Кожна комірка масиву H є вказівником на зв'язаний список (ланцюжок) пар ключ-значення, відповідних одному і тому самому хеш-значенню ключа. Колізії просто призводять до того, що з'являються ланцюжки довжиною більше одного елемента. Операції пошуку або видалення елемента вимагають перегляду всіх елементів відповідного ланцюжка, щоб знайти в ньому елемент з заданим ключем. Для додавання нового елемента необхідно додати елемент в кінець або початок відповідного списку, і, у випадку якщо коефіцієнт заповнення стане занадто великим, збільшити розмір масиву H і перебудувати таблицю.

2) Як буде виглядати хеш-таблиця з ланцюжками після того, як в неї послідовно помістили елементи з ключами (див. таблицю, стовпець 2) (в зазначеному порядку)? Число позицій в таблиці дорівнює 9, хеш-функція має вигляд h(k) = k mod 9.

Для хеш таблиці ланцюжками я створив новий класс HashTableWithChains з відповідними потрібними методами додавання, видалення, пошуку елементів та роботою з файлом.

## Блок схеми методів

Функція додавання ключа

void HashTableWithChains::Add(int key)

{

int hash = HashFoo(key);

if (node[hash] == nullptr)

{

node[hash] = Create(key);

}

else {

Node\* temp = node[hash];

while (temp->next != nullptr)

{

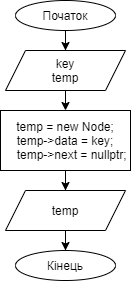
temp = temp->next;

}

temp->next = Create(key);

}

}

Функція створення нового ланцюжка

HashTableWithChains::Node\* HashTableWithChains::Create(int key)

{

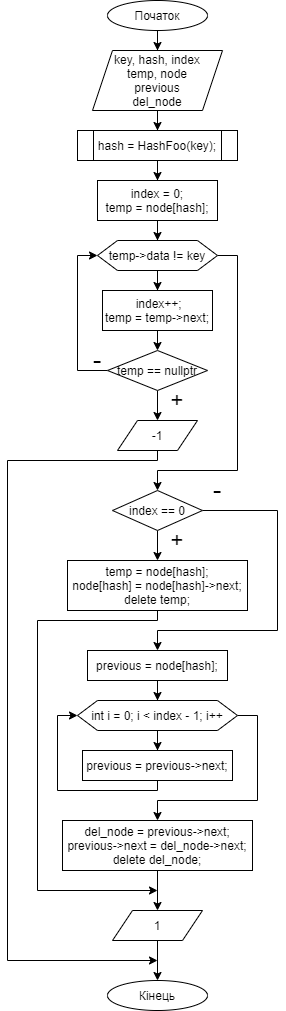
Node\* temp = new Node;

temp->data = key;

temp->next = nullptr;

return temp;

}

int HashTableWithChains::Delete(int key)

{

int hash = HashFoo(key);

int index = 0;

Node\* temp = node[hash];

while (temp->data != key)

{

index++;

temp = temp->next;

if (temp == nullptr)

return -1;

}

if (index == 0)

{

temp = node[hash];

node[hash] = node[hash]->next;

delete temp;

}

else {

Node\* previous = node[hash];

for (int i = 0; i < index - 1; i++)

{

previous = previous->next;

}

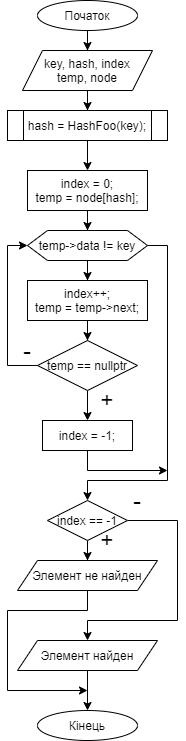
Node\* del\_node = previous->next;

previous->next = del\_node->next;

delete del\_node;

}

return 1;}

Функція пошуку ключа

void HashTableWithChains::Search(int key)

{

int hash = HashFoo(key);

int index = 0;

Node\* temp = node[hash];

while (temp->data != key)

{

index++;

temp = temp->next;

if (temp == nullptr) {

index = -1;

break;

}

}

if (index == -1)

{

cout << endl << "Элемент не найден" << endl;

}

else {

cout << endl << "Элемент найден в строке " << hash

<< " под индексом " << index << endl;

}

}

## Робота програми

Рис. 2 – занесені ключі з варіанту та пошук елемента

## Аналіз

При припущенні, що кожний елемент може потрапити в будь-яку позицію таблиці H з однаковою ймовірністю і незалежно від того, куди потрапив будь-який елемент, пересічний час роботи операції пошуку елемента складає *Θ(1 + α)<i>, де <i>α* — коефіцієнт заповнення таблиці.

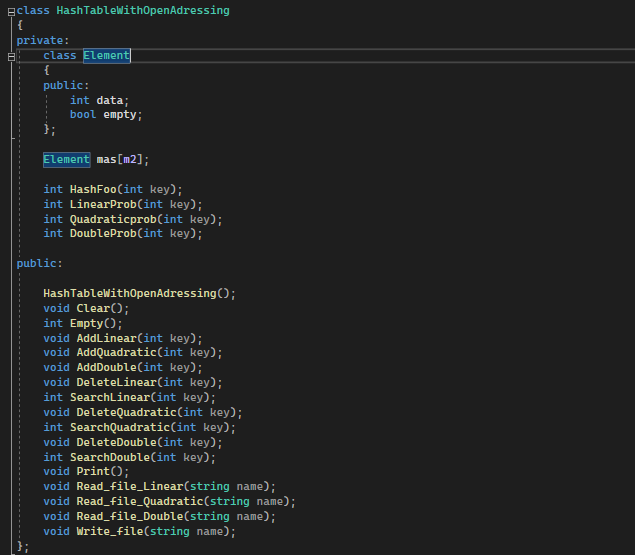
Складність:

* Найгірший випадок – ;
* Найкращий випадок – ;
* Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

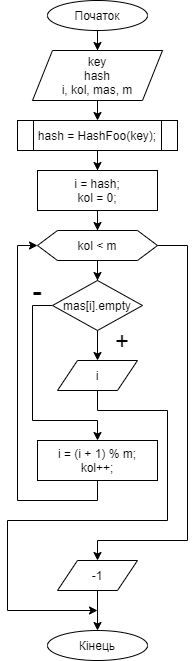
# Хеш таблиця з відкритою адресацією

В стратегії, названій відкритою адресацією, всі записи зберігаються в самому масиві H. Коли додається новий запис, масив перевіряється в певному порядку доки не буде знайдена вільна комірка куди й вставиться елемент. У випадку пошуку елемента, масив сканується в тій самій послідовності доки потрібний запис або порожня комірка не буде знайдена, друге означає відсутність елемента. Назва «відкрита адресація» показує, що розташування («адреса») елемента не визначається його хеш-значенням. Цей метод також називають закритим хешуванням. В загальному випадку, послідовність в якій переглядаються комірки хеш-таблиці залежить тільки від ключа елемента, тобто це послідовність h0(x), h1(x), …, hn-1(x), де x — ключ елемента, а hi(x) — довільна функція, яка зіставляє кожен ключ комірки з хеш-таблицею.

3) Виконайте додавання ключів (в зазначеному порядку, див. таблицю стовпець 2) в хеш-таблицю з відкритою адресацією розміру m = 11. Для обчислення послідовності проб використовується лінійний метод з h'(k) = k mod m. Виконайте те ж завдання, якщо використовується квадратичний метод з тієї ж h', c1 = 1, c2 = 3, а також для подвійного хешування з h1 = h' и h2(k) = 1 + (k mod (m - 1)) .

Для таблиці з відкритою адресацією було створено новий класс HashTableWithOpenAdressing з відповідними методами додавання, видалення, пошуку елементів та роботою з файлом.

## Блок схеми методів

Лінійне пробування

int HashTableWithOpenAdressing::LinearProb(int key)

{

int hash = HashFoo(key);

int i = hash;

int kol = 0;

while (kol < m2)

{

if (mas[i].empty)

return i;

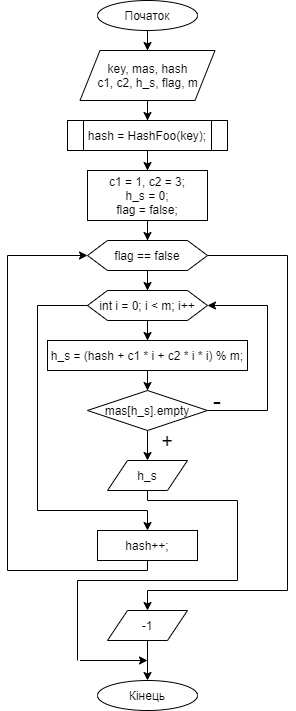
i = (i + 1) % m2;

kol++;

}

return -1;

}

Квадратичне пробування

int HashTableWithOpenAdressing::Quadraticprob(int key)

{

int c1 = 1;

int c2 = 3;

int hash = HashFoo(key);

int h\_s = 0;

bool flag = false;

while (flag == false)

{

for (size\_t i = 0; i < m2; i++)

{

h\_s = (hash + c1 \* i + c2 \* i \* i) % m2;

if (mas[h\_s].empty)

{

return h\_s;

}

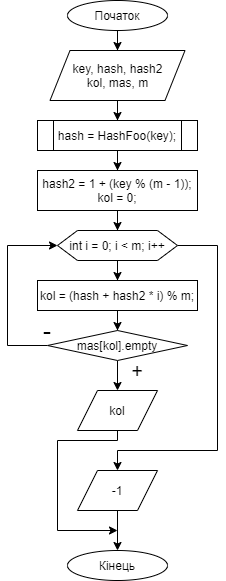
}

hash++;

}

return -1;

}

Подвійне пробування

int HashTableWithOpenAdressing::DoubleProb(int key)

{

int hash = HashFoo(key);

int hash2 = 1 + (key % (m2 - 1));

int kol = 0;

for (size\_t i = 0; i < m2; i++)

{

kol = (hash + hash2 \* i) % m2;

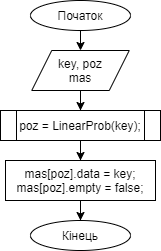
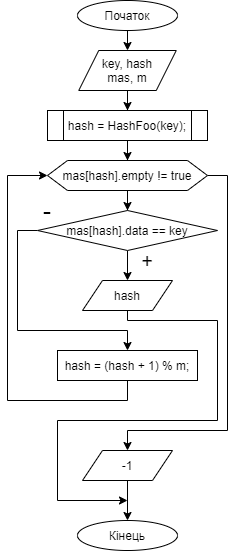
if (mas[kol].empty)

return kol;

}

return -1;

}

Функція додавання та пошуку ключа

void HashTableWithOpenAdressing::AddLinear(int key)

{

int poz = LinearProb(key);

mas[poz].data = key;

mas[poz].empty = false;

}

int HashTableWithOpenAdressing::SearchLinear(int key)

{

int hash = HashFoo(key);

while (mas[hash].empty != true)

{

if (mas[hash].data == key)

{

return hash;

}

hash = (hash + 1) % m2;

}

return -1;

}

## Робота програми

Рис. 3 – Лінійне

Рис. 4 – Квадратичне

Рис. 5 – Подвійне

Аналіз

Лінійне пробування

* Найгірший випадок – ;
* Найкращий випадок – ;

– Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

Квадратичне пробування

* Найгірший випадок – ;
* Найкращий випадок – ;

– Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

Подвійне хешування

* Найгірший випадок – ;
* Найкращий випадок – ;

– Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

# Висновки

Існує два типи хеш таблиць:

* З ланцюжками;
* З відкритою адресацією.

Різницею між цими типами хеш-таблиць є те, що масив *H*, що використовується для збереження даних, містить пари значень(з відкритою адресацією), або ж списки пар(з ланцюжками).

Виконання операцій в хеш-таблиці починається з обчислення хеш-функції від ключа. Отримане хеш-значення *i = hash(key)* відіграє роль індексу в масиві *H*. Після цього операція (додавання, видалення, пошук) перенаправляється об'єктові, який зберігається у відповідній комірці масиву *H[i]*. Ситуація, коли для різних ключів отримується одне й те саме хеш-значення, називається колізією. Такі події непоодинокі — наприклад, при додаванні в хеш-таблицю розміром 365 комірок, усього лише 23-х елементів, ймовірність колізії вже перевищує 50 відсотків (якщо кожний елемент може з однаковою ймовірністю потрапити в будь-яку комірку). Через це механізм розв'язання колізій — важлива складова будь-якої хеш-таблиці.

Властивості хеш-таблиці.

Важлива властивість хеш-таблиць полягає в тому, що при деяких розумних припущеннях, всі три операції (пошук, вставлення, видалення елементів) зазвичай виконується за час O(1). Але при цьому не гарантується, що час виконання окремої операції малий. Це пов'язано з тим, що при досягненні деякого значення коефіцієнта заповнення необхідно здійснювати перебудову індексу хеш-таблиці: збільшити розміри масиву H і заново додати в порожню хеш-таблицю всі пари.